Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2016) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

25th International Meshing Roundtable

A Marching-Tetrahedra Algorithm for Feature-Preserving Meshing of Piecewise-Smooth Implicit Surfaces

Brigham Bagley[a], Shankar P. Sastry[a], Ross T. Whitaker[a]

[a]*Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112*

Abstract

For visualization and finite element mesh generation, feature-preserving meshing of piecewise-smooth implicit surfaces has been a challenge since the marching cubes technique was introduced in the 1980s. Such tessellation-based techniques have been used with varying degrees of success for this purpose, but they have consistently failed to reproduce smooth curves of surface-surface intersection when two surfaces intersect at sharp angles. Such techniques attempt to discretize all surfaces within a given cell in a single pass by computing edge-surface points of intersection for each edge in the cell and use predefined stencils to generate the surface mesh elements. This approach limits the number of surface-edge intersections on every edge to just one (or some small finite number) because the number of stencils grows exponentially with the number of surfaces. In our tessellation-based approach, we discretize only one surface in each pass over the tetrahedral cells and retetrahedralize the affected cells for the next surface during the next pass. As a result, we manage to preserve sharp features in the domain, and our algorithm scales almost linearly with the number of surfaces. As in the isosurface-stuffing algorithm, we locally warp the initial tessellated domain to ensure that a high-quality surface mesh is generated.

Peer-review under responsibility of the organizing committee of IMR 25.

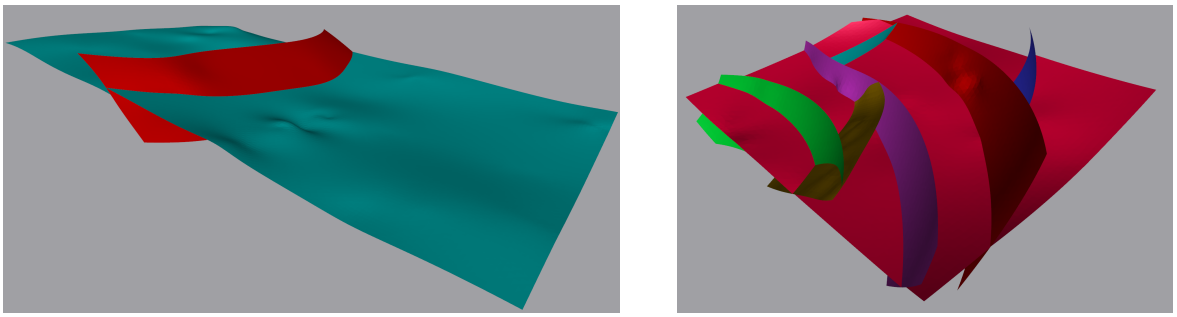
Keywords: marching tetrahedra, feature preserving, surface meshing;

1. Introduction

The generation of surface meshes for implicitly defined surfaces has been studied over the last few decades [21] due to the applications of the meshes in areas such as biomedical simulations (from patient images) [9], constructive solid geometry (CSG) [5], and geometric modeling and visualization of complex real-world shapes (statues, for example) [17]. Our motivation to develop an algorithm for this purpose arose from our need for geometric modeling and simulation of subsurface geology. The primary challenge is to mesh our implicit functions quickly and preserve sharp features where two surfaces or more intersect. In subsurface geology, a horizon is a layer of soil parallel to the crust whose physical characteristics slightly differ from neighboring layers. There are discontinuities in horizons caused by massive tectonic forces during earthquakes or other geological events. Such meshes are necessary for analyzing

* Brigham Bagley eMail: brig@sci.utah.edu

1877-7058
Peer-review under responsibility of the organizing committee of IMR 25.



(a) A Simple network (b) A Complex Network

Fig. 1. Examples of typical domain seen in structural geology. We wish to capture the curve of intersection between two surfaces in our surface meshes. Existing techniques have been unable to do that when the intersection is "sharp".

water tables, extracting natural resources, analyzing impact of an earthquake, etc. Another example of such domains is layered, composite materials in semiconductor industries, where multiple materials are deposited layer by layer for manufacturing a computer chip. In this paper, we describe an algorithm to generate a surface mesh for geological structures using implicit functions whose level sets define the surface. The algorithm is generic enough to be used in other applications such as CSG.

The nature of the domains we wish to mesh is shown in Fig. 1. In the first example, we have one horizon split by a fault. The horizon surfaces on the either side of the fault are not continuous. In the second example, a complicated geological structure is shown. There are multiple faults dividing a single horizon into many different compartments, each having their own equations defining the surface.

Our technique builds upon prior tessellation-based algorithms such as the marching cubes [21], isosurface stuffing [19], and dual contouring techniques [17,20]. Such techniques build a surface mesh by first constructing a background mesh and then determining the edges though which the surfaces pass. These techniques preserve sharp features with difficulty because they restrict the number of surfaces passing through an edge to just one. As a result, some approximations are made to the geometry that result in elimination of sharp features. It is possible to extend these algorithms for preservation of sharp features by allowing multiple surfaces to cross an edge, but the combinatorial explosion in the number of possible configuration makes it very hard to implement them. Some details about these algorithms are provided in Section 2. In the techniques above, all the surfaces are resolved simultaneously in a single pass over the tetrahedral or the cubical elements in the background mesh.

Our technique is based on the isosurface-stuffing algorithm [19], which was developed for a single surface. We extend the technique for multiple surfaces, and show that it is possible to preserve sharp features in the domain at the cost of mesh quality. As in the isosurface-stuffing algorithm, we employ a tetrahedral warping strategy to retain higher quality. Section 3 provides a brief description of the isosurface stuffing algorithm.

Our algorithm overcomes the difficulty of preserving features by constructing each surface sequentially and subdividing each affected tetrahedron before constructing the mesh for the next surface. Although we make multiple passes over the set of tetrahedral elements, only a single surface is consolidated to the mesh, and we are able to preserve sharp features, in contrast to other techniques thus far. A detailed explanation of our algorithm is provided in Section 4. As expected from tessellation-based techniques (see [19]), our implementation is fast, and (depending on the material intersections) our meshes are of good quality (explained in detail in Section 5). In order to further improve the quality of the triangles in the surface mesh, we use a smart Laplacian-based smoothing technique to move the vertices around.

Our algorithm is defined for implicit surface functions and will need extensions for application in the biomedical field, where several scalar fields provide different materials in the domain rather than an implicit surface. We briefly discuss the possible changes in the algorithm and other future research directions in Section 6.

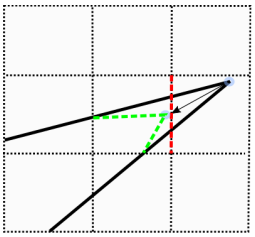


Fig. 2. The dotted black lines are part of the background mesh, and the solid black lines are the surfaces that need to be meshed. On the red line, there are two edge-cuts. Most algorithms do not have stencils that can handle multiple cuts, and they project the point of intersection into one of the elements. As a result, when meshing in 3D, these points align themselves erroneously in a zigzag pattern. The resulting discretization is shown in green.

2. Related Work

As reported in [16], attempts to mesh implicit surfaces may be broadly classified as tessellation-, Delaunay refinement-, Morse's theory-, and surface tracking-based techniques. We provide a brief description of some the techniques and mention the relative advantages and disadvantages of using such algorithms.

2.1. Tessellation-Based Technique

The marching cubes algorithm [21] is one of the earliest examples of tessellation-based algorithms. With these algorithms [4,8,19,21], a background grid is trivially constructed using octtree-based decomposition techniques. Along each edge, the surface-edge crossing points are computed and a polygonal mesh is constructing using predefined stencils. For improved adaptability, the background may be constructed using sophisticated techniques such as feature-size computation [9]. An advantage of the tessellation-based algorithms is that they are very fast because the surface-edge intersection may be easily computed using the iterative bisection algorithm, and they can also be easily implemented in parallel.

As with other algorithms of this type, we may produce an overrefined mesh if the background mesh is highly refined. At the cost of greater computation time to estimate the local feature size, we can lower the size of the mesh by constructing an appropriate background mesh.

Dual contouring techniques [17,18,20,27] are capable of capturing sharp features as long as the dihedral angle at the crease is not too small. In these techniques, the value of the implicit function as well as of the Hermite data (first-order derivatives) is used to locate the points at which implicit functions meet. The location is estimated by optimizing an error function that accounts for the consistency with the Hermite data provided to the algorithm. The location of the points at which the surfaces intersect is approximated by optimizing an error function.

A drawback of these algorithms is that they assume just one surface crossing along an edge even in the case of multiple materials because it is complicated to construct the predefined stencils for a greater number of intersections. In addition, the location of the intersection of multiple surfaces is constrained to within an element where two (or more) surfaces intersect the element at some point on the respective edges. When two surfaces intersect sharply along a curve, it is very likely that the two surfaces pass through a subset of edges. In this case, these algorithms have mechanisms to project the points on the curve of the intersection such that only one surface-edge intersection is present along all edges (see Fig. 2). These project techniques result in aliasing of the curve of the surface-surface intersection in the final mesh, where the curve follows a zig-zag pattern.

2.2. Delaunay Refinement-Based Technique

With Delaunay refinement-based techniques [2,7,10–12,15,23], the dual of the restricted Voronoi diagram is used to mesh the surface. The main advantage of these techniques is the guarantee of the topology of the input surface and the resulting output mesh under the constraints of sufficient sampling. The sampling constraints are directly related to the local feature size at various points on the surface. The local feature size is difficult to compute, so these techniques assume a certain value and bootstrap the mesh construction. These algorithms also assume that the curve

of the intersection of two surfaces is provided as input by the user. The computation of the restricted Voronoi diagram is more expensive than computing the location of the edge-surface intersection; thus, these algorithms are slower than tessellation-based algorithms.

2.3. Morse Theory-Based Technique

Morse theory has been used to design and prove algorithms that guarantee the homeomorphism between the implicit surface and the constructed mesh under certain constraints [6,26]. Many of the Delaunay refinement-based algorithms have been proven to preserve the topology based on Morse's theory.

2.4. Surface Tracking-Based Technique

Surface tracking algorithms [1,3,24,25] typically assume that the gradient and the Hessian of the implicit function defining the surface are known or easy to compute. These algorithms use that information to smartly place points on the surface and construct a mesh. These techniques provide a small mesh at the cost of computation time, but some of the components in a multicomponent domain may not be meshed if the bootstrapping seed points are not chosen carefully. Some algorithms use variable techniques to seed vertices and move them around to improve the quality of the elements [14,22,28]. With these techniques, it is not always possible to keep the vertices exactly on the surface. To remedy this, an additional step projects the vertices back onto the actual surface [16].

3. Background

We will first discuss what implicit functions are, and then, since our algorithm is an extension of Labelle and Shewchuk's isosurface stuffing algorithm [19], we review their method.

Given a function $f(x)$ and a constant k, where x is a 3D vector, an implicit surface is a level set where $f(x) = k$. An explicit surface has the form $z = f(x)$, where x is a 2D vector. The input to our algorithm is a set of functions and a set of constants. Inside the surface, the value of the function is greater than 0, and it is less than zero outside the surface (or vice versa). In our implementation, some additional tree-based data structure is used to denote the relationship between the surfaces, i.e., if surface $f_1(x) = k_1$ is on the positive side of surface $f_0(x) = k_2$, surface 0 is the right child of surface 1. This tree is also a part of the input.

In the isosurface stuffing algorithm, only a single surface is considered. We extend their work for multiple surfaces. The isosurface-stuffing algorithm begins with a uniform background mesh that is a body-centered cubic lattice. A Delaunay tetrahedralization of the lattice provides the background mesh necessary for volume meshing. The input to the algorithm is given in the form of an implicit function that is positive inside the object being meshed and negative outside. Naturally, the vanishing level set is the surface of the object. The value of the implicit function is computed at each vertex on the background mesh. On each edge of the mesh, if the implicit function changes sign, the edge-cut location is marked. It is assumed that only one edge-cut is present along an edge. Based on the edge-cut locations along each edge of a tetrahedron, one of the many standardized stencils is applied to appropriately mesh the domain. In addition, if the edge-cuts are too close to an existing vertex, the vertex is moved to that location so that the resulting tetrahedra have good quality. In the process, the tetrahedra in the background mesh are being cut and/or warped. Because the warping operation is carried out when the edge-cuts are very close and the cut operation happens away from the existing vertices, the quality deterioration of the tetrahedra is bounded. Thus, it is possible to provide bounds on the dihedral angles in the resulting final mesh.

4. Algorithm

Our algorithm is a tessellation-based technique, but we mesh a single surface at a time and retetrahedralize the split elements in the background mesh before we attempt to mesh the second surface. Our technique captures the sharp features present in the geometry. Since the mesh is distorted after the insertion of a surface, the quality of the background tetrahedra deteriorates near the curves of surface-surface intersection with every new surface. Unlike the isosurface stuffing algorithm, however, we cannot explicitly guarantee the quality of the output surface meshes. This

is because it is possible to construct adversarial example where two surface meet at really small angles along the curve of intersection. The warping operation is forbidden for a vertex on an existing surface and cannot be warped to a new surface. As a result, poor-quality triangles are common. Those cases can be eliminated by moving the vertex for the new surface mesh element, but keeping the existing surface mesh intact in the memory. This allows a user to construct a high-quality surface mesh, but the underlying volume mesh is lost. As we wanted the option of keeping the volume mesh, we decided to not pursue this route. In practice, we obtain meshes with good quality without this virtual vertex technique, but using the Laplacian smoothing technique instead. We leave the computation of the bounds on the angles of the triangle with the use of the virtual vertex technique for future research.

We first provide a brief description of the algorithm and explain how it performs better than existing techniques. We then describe each step in greater detail. The input to the algorithm is a set of implicit functions representing the surfaces we wish to mesh. If needed, an additional data structure may be used to denote requirements such as meshing a surface only on the positive (or negative) side of another surface, etc.

Our algorithm to construct the surface mesh is as follows:

1. Construct a background tetrahedral mesh.
2. Consider the first surface and determine the locations where the implicit function vanished along those edges of a tetrahedron on which the function changes sign.
3. Warp and/or cut the elements depending on the proximity of the edge-cut to the vertex. Move vertices that already belong to the surface along the line of intersection with the current surface.
4. Retetrahedralize the affected tetrahedra, and repeat steps 2 and 3 for the next surface.
5. Move the vertices along their respective surfaces to optimize the mesh quality.

First, we construct a homogeneous, structured mesh by placing vertices on a Cartesian grid. We construct cubes on the Cartesian grid and divide each cube into five tetrahedra as shown in Fig 3 . Our choice was motivated by our need for surface meshes, not volume (adaptive or nonadaptive) meshes. Our technique also provides a high-quality volume mesh. If the readers wish to implement the algorithm for obtaining volume meshes, a BCC lattice-based mesh is probably a better choice as attempted in [19] and [8] because it has higher minimum dihedral angles. Additionally, users may also use their own background mesh based on their requirements, as has been done in [9].

Second, we determine the location of the edge-cuts using a simple bisection technique, as it is robust to numeric precision to conform to geometry. We found that Newton's technique is unstable in many cases. We did not appropriate the implicit function to be linear inside an element because we wanted to reproduce the geometry as close as possible to the implicit surfaces.

Third, we warp and/or cut the elements based on the edge-cut locations. Our algorithm is identical to the isosurface-stuffing algorithm [19] for a single surface. More details on the algorithms may be found in the original paper. In our implementation, we use the ratio of distance of the edge-cut to the location of the vertex and total length of the edge to determine if the tetrahedron needs to be warped. If the ratio is smaller than a certain $\alpha = 0.2$, we move the vertex to the edge-cut. Note that as we move the vertex to the edge-cut, other edge-cuts on the edges adjacent to the vertex disappear because we assume that only one edge-cut (or zero) may be present on an edge for each surface. Thus, it is important to cut the elements only after we have warped all the affected tetrahedra and updated the edge-cut locations. The edge-cuts may be classified as shown in Fig. 4. If a vertex is already part of a surface, we allow the vertex to move only if it will be warped along the element of the corresponding surface mesh. If the warp will move existing geometry elements, the operation of the warp is denied, which may result in poor quality triangles when two surfaces come arbitrarily close to each other. In order to avoid creation of such elements, a refined or an adaptively refined background mesh or constrained geometry warping may be used. One can always construct adversarial surfaces meeting at small angles so that the meshes have some poor-quality triangles near the curve of the surface-surface intersection. In order to eliminate this artifact completely, one can carry out the warping operation with moving any vertex on an existing surface, but "virtually" moving the same for the purpose of meshing the new surface. This change will result in good-quality elements, but for our purpose, we found that the use of the smart Laplacian technique below fetched good results.

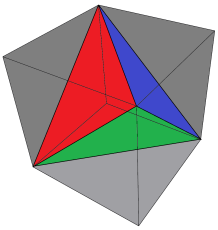
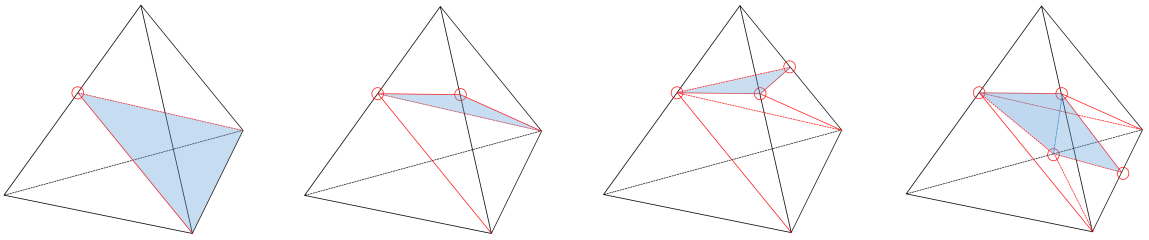


Fig. 3. Decomposition of a cube in the background mesh into five tetrahedra. We use a uniform background mesh. If an adaptive octtree is used, the decomposition has to be different to account for hanging vertices and edges. Instead of this decomposition, the implementation may also use the BCC lattice.



(a) One Edge-Cut (b) Two Edge-Cuts (c) Three Edge-Cuts (d) Four Edge-Cuts

Fig. 4. Four possible stencils for a surface intersecting a tetrahedron. If a surface passes through a vertex, it is not considered an edge-cut. Depending on how a surface intersects a tetrahedron, these stencils are used to construct the surface mesh. In addition, trivial cases where the surface insects one, two, or three vertices are also handled appropriately. The case where the surface passes through all four vertices is ambiguous, so we ignore it in our implementation. Care was taken to ensure that the stenciling was consistent across the mesh so that we do not end up with hanging edges or topological holes in the mesh.

Fourth, we retetrahedralize the affected tetrahedron and repeat the above steps. If a high-quality volume mesh is needed, it may be prudent to optimize the mesh at this stage, but for our purpose of generating a surface mesh, we found that it was not necessary. We then repeat the above two steps for the next surface in the input.

Fifth, we use a smart Laplacian-based smoothing system to improve the quality of the mesh. This step is required only if the generated mesh is of poor quality. In our experiments, we found that relatively poor-quality elements are found only in regions where two surfaces are arbitrarily close to each other. In such cases, some of the vertices may not warp to the edge-cuts because they belong to an existing surface. After generation of the surface mesh, those edge-cuts become vertices, and we move the vertices to improve the quality of the mesh. In this technique, we compute the average vector from a vertex to all its adjacent vertices on the surface. If the vertex under consideration is on the curve of the intersection of two surfaces, only the neighboring vertices on the curve are considered as adjacent vertices. If it improves the quality of the triangles (increases the minimum angle), we move the vertex in the direction of the average vector by a distance that is equal to a certain factor (we used 10%) of the average magnitude of the vectors. We then project the vertex back onto the surface (if it is on one surface). The simple algorithm was suitable for our application. As many iteration of the smoothing technique as needed may be performed until the quality of the mesh stops improving. If higher-quality meshes are desired, numerical optimization algorithms may be used for this step.

4.1. Mesh Quality - A Discussion

The quality of the surface mesh is affected by the warping and cutting operations. If the edge cuts are close to an existing vertex, the vertex is warped. If not, the tetrahedron is cut. When the vertex is warped, it is moved by a small distance. Thus, the quality deterioration is bounded. When the tetrahedron is cut, the new vertices are all sufficiently far from the existing ones, but it is within the tetrahedron. Thus, the quality of the triangles created is not allowed to be far worse than existing triangles in the mesh. In an adversarial case, some elements can undergo both warping and cutting. We may compute the bound on the quality of the mesh elements created by our algorithm in such cases, and the bound is a function of the initial mesh. This has been clearly shown in [19].

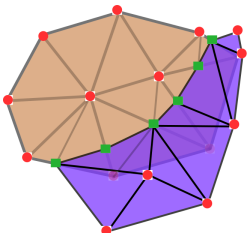


Fig. 5. Two surface meshes intersect along a polyline as shown. The orange circle denote the vertices that are exclusively on either of the two surfaces. The green squares denote the vertices that are present on both surfaces. In our method, we use the bisection technique to compute the location of the orange vertices. The green vertices are found by computing the intersection of two triangles. Thus, they do not exactly lie on the actual surface defined by the implicit function.

When two surfaces interact, the warping operation may be carried out in all cases either virtually (as described above) or along the surface, and the cutting operation is identical. Thus, the quality deterioration is still bounded, but this deterioration takes place from an already-distorted mesh. Therefore, the bound on quality is lower.

When three surfaces interact, an existing triangle on the surface mesh can get cut by two or more surfaces that are very close. In such cases, there is no bound on the reduction in the quality of the existing triangle as we shall see in section below in the Kutum and Dee datasets. In those cases, a refined background grid is the only choice, where the element size is proportional to the local feature size. Thus, the quality bound is also proportional to the ratio of the local feature size and the element size. When multiple surfaces intersect, the feature size goes to zero, and the minimum angle bound is constrained by the angles at which the three surfaces intersect. For instance, if two surfaces intersect sharply at a point and the third surface intersects both surface at that point, an element on the mesh of the third surface at the point on concurrence of the three surfaces will have a small angle.

4.2. Merits and Demerits

Our technique only needs the equation of the surfaces, but not its gradient or Hessian as it is required in dual contouring techniques. The curves of intersection are not necessary either as in the case of Delaunay-based techniques, where the curves have to be explicitly provided. This makes our technique very fast.

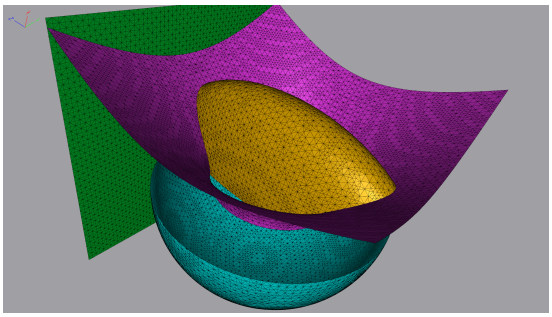
A disadvantage of the technique is that the mesh vertices on the curve of intersection are not exactly on the curve of intersection. We are effectively intersecting two surface meshes of the smooth surfaces. Thus, the vertices of the poly line at which they intersect may not lie exactly on the curve of intersection. The distance between the mesh vertex and the closet point on the curve is bounded by maximum distance between a point on a mesh edge and the surface (see Fig. 5). This algorithm does not capture small features surrounded by just one surface unless the background grid is fine enough. No other algorithm currently can succeed in doing that.

5. Results and Discussion

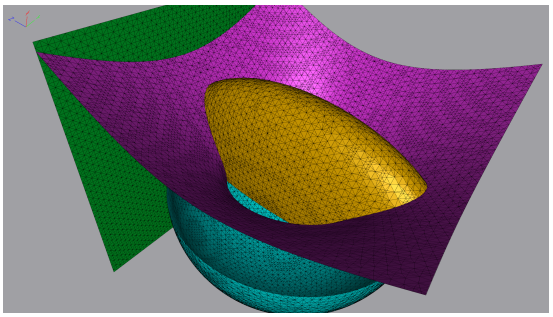
We implemented the algorithm in C++ and used Qt libraries to visualize of the meshes we generated. The implementation was tested using several synthetic datasets consisting of planes, spheres, cylinders, ellipsoids, etc. intersecting in various parts of the domain. We then used our implementation on realistic geometry from subsurface geology to construct meshes of faults and horizons. These domains show the extent of success and the limitations of this technique to generate high-quality meshes. We generated meshes for the following five different domains: shapes, saddle, cone and geological structures Dee and Kutum. The first three examples are synthetic and the last two are inspired by real-world domains. In our result we have also provided statistical details about angles of the triangles in the mesh produced by our algorithm before and after smoothing. The inputs to our algorithm include the functions defining the surfaces, the resolution of the uniform background grid, and the number of iterations of the “smart” Laplacian-based smoothing technique. Table 1 provides the details about the size of the background grid, surface mesh size and other details about the meshes obtained from our technique.

domain	grid	# elements	time		minimum angle	
			meshing	smoothing	no smoothing	smoothing
shapes	60*60*61	83,737	7.96	6.39	4.40	11.52
saddle	60*60*61	76,665	5.60	6.39	1.47	2.59
cone	60*60*61	61,840	5.59	4.30	9.65	18.64
Kutum	147*95*15	137,721	35.62	22.29	0.54	2.92
Dee	125*98*53	203,977	216.77	69.04	0.11	0.13

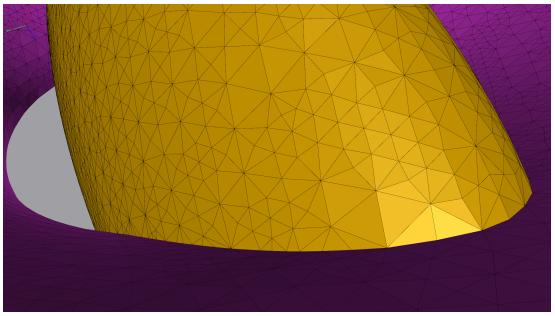
Table 1. Details about the meshes generated for the domains described in the paper. The time taken to generate the mesh is roughly proportional to the size of the background grid. The smoothing time is roughly proportional to the number of elements in the surface mesh. The minimum angle improves with smoothing. The small angles in the Kutum and Dee datasets are due to two surfaces being very close to each other and intersected by a third surface. The triangles on the third surface in between the two surfaces are sharp unless the background grid is very fine.



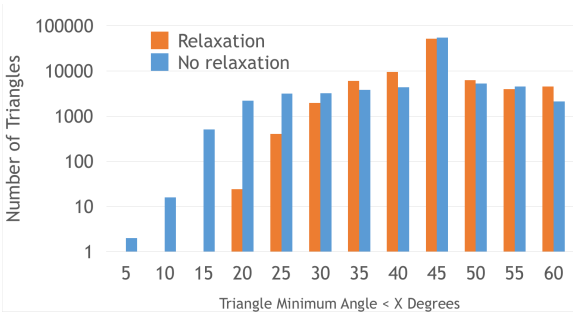
(a) Mesh Before Smoothing



(b) Mesh After Smoothing



(c) Zoomed-in View



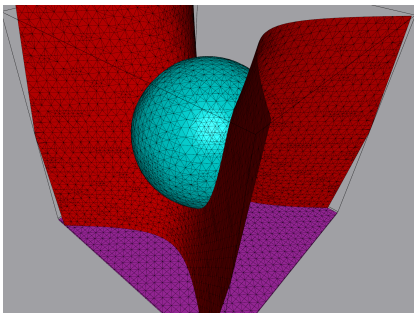
(d) Angle Histogram

Fig. 6. Results from the shapes domain. Ten iterations of smoothing were performed.

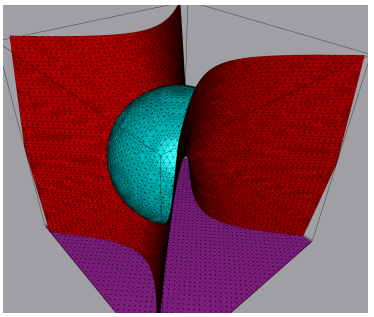
The shapes domain is shown in Fig. 6. It consists of a sphere, an ellipsoid and a hyperboloid. The zoomed-in view of the curve of intersection is also shown. The readers can clearly see that the curve is smooth without any aliasing effects. We have provided the histogram of angles in the triangular elements in Fig. 6(d).

The saddle domain is show in Fig. 7. It contains a sphere and a saddle. A characteristic feature of tessellation-based techniques can be seen on this domain as shown in Fig. 7(d). As the feature size are relatively small at the saddle point and the surface is oriented is a direction that is not parallel to the Cartesian grid, there are anisotropic elements near the saddle point. If an unstructured mesh is used as a background mesh, these anisotropic feature will disappear. As in the previous example we have also provided a close-up of the curve of the intersection and the histogram of angle in the mesh.

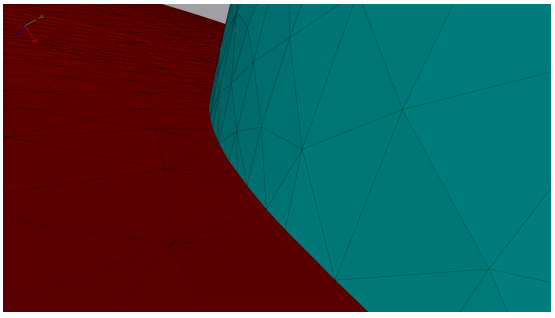
The cone domain was meshed as shown in Fig. 8. The domain consists of a cone and a part of a sphere. An artifact of the our technique (and other marching cubes/tetrahedra techniques) is the loss of small features The feature size of a cone goes to zero at the “apex” of the cone. Thus, we lose parts of the feature near the apex as show in Fig. 8(c). In order to capture such features, we need a finer background mesh. In addition, incorporation of interval or affine



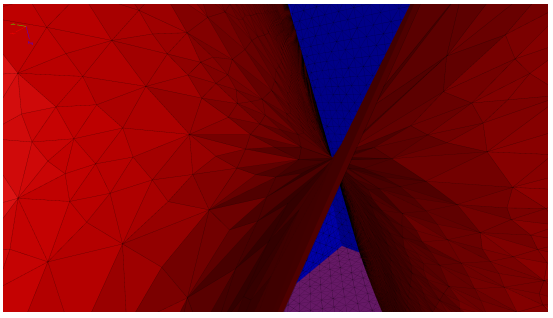
(a) Mesh Before Smoothing



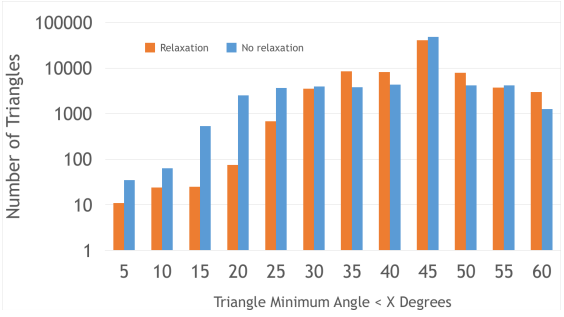
(b) Mesh After Smoothing



(c) Zoomed-in View



(d) Zoomed-in View



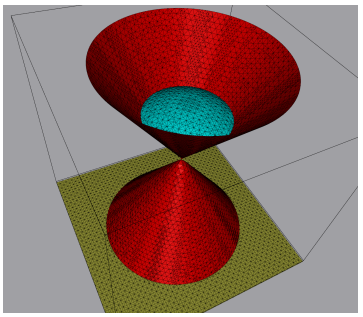
(e) Angle Histogram

Fig. 7. Results from the saddle domain. Two zoomed-in views are present to show the smooth curve of the intersection and anisotropic triangles near the saddle point. As in the previous case, ten iterations of smoothing were performed.

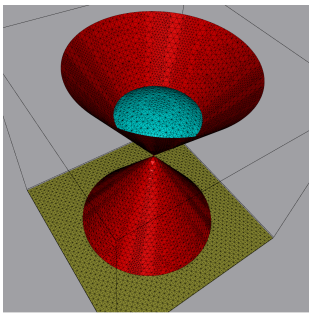
arithmetic techniques is necessary so that we do not lose any potential edge-cut. In our case, we captured the apex because was on an existing grid point (we were lucky), but we did miss other features close to the apex.

The faults and horizon from the Kutum domain is shown in Fig. 9. The surfaces shown are interpolated from radial basis functions constructed from point sets. Domains such as these were the real purpose for the development of our technique. For numerical simulation, it is important to preserve the sharp features at the curves of intersection. The meshing of this domain is relatively easy since these surfaces are close to planar and the horizons are well separated from each other on the either side of the fault. We have shown the discretized curve of the intersection to illustrate how warping enables us to retain the quality of the background mesh in the final mesh. Note the lack of triangles with tiny angles at the interface. On the surface mesh, you can also see a “curve” of edges right above the curve of intersection of the two surface meshes. This curve is due to the intersection of the horizon surface on the other side of the fault.

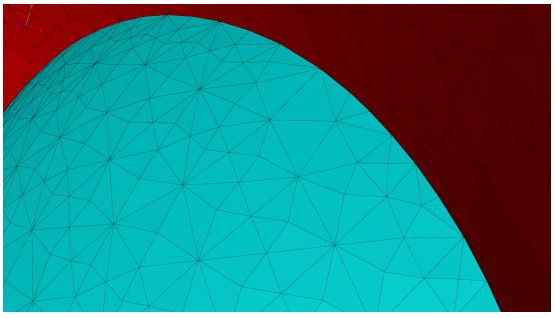
We now move on to a more difficult domain to mesh. The surface mesh from the Dee domain is shown in Fig. 10. This domain contains two horizons that are very close to each other. Due to such a challenging geometry, it is not



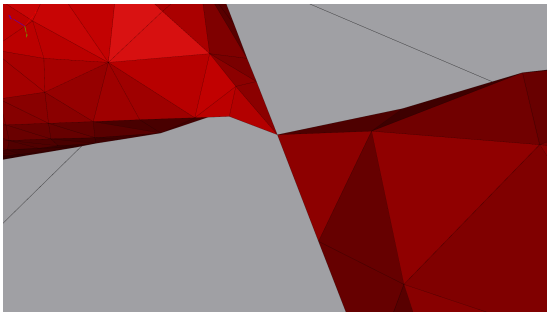
(a) Mesh Before Smoothing



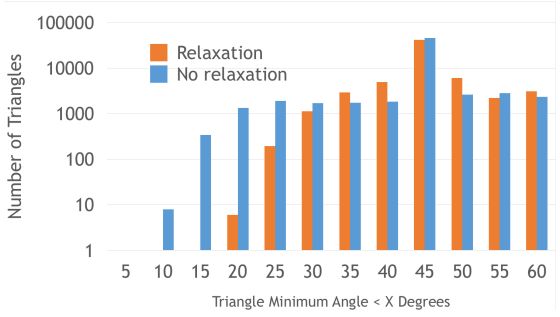
(b) Mesh After Smoothing



(c) Zoomed-in View



(d) Zoomed-in View



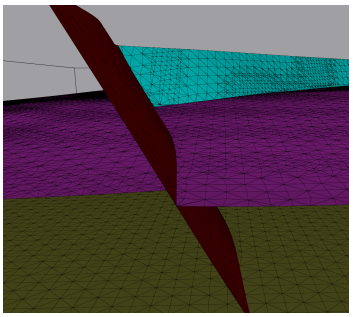
(e) Angle Histogram

Fig. 8. Results from the cone domain. Ten iterations of smoothing were performed. The curve of the intersection is smooth, but the technique cannot capture all the small features present in just one surface, which is an inherent drawback of this technique.

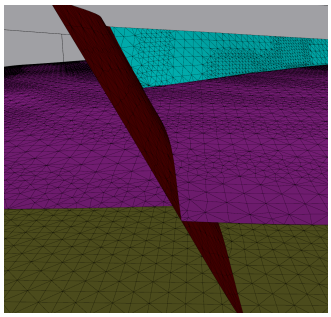
always possible to carry out the warping operation to produce high-quality meshes. For most elements, it is possible to get rid of bad elements by refining the background mesh, but it results in a large mesh. In this domain, we wanted to produce a coarse mesh of good quality. That is possible by using a coarse background mesh and smoothing the vertices. The figure shows the mesh near the intersection before and after ten iterations of smoothing. As in the Kutum mesh, you can see a curve (that is discretized by edges) due to horizon surfaces intersecting on the other side of the fault.

6. Conclusions and Future Work

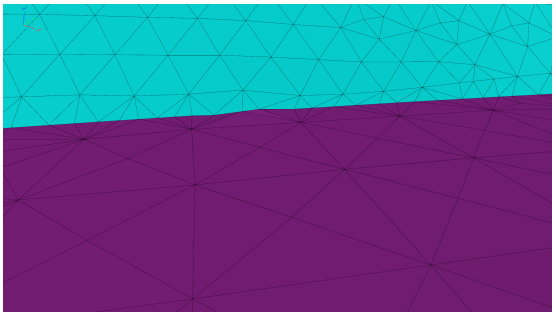
Reproducing the curve of intersection though a mesh is a long-standing problem in the field of meshing and visualization. We provided a solution to the problem by observing that the earlier techniques made just one pass on the tetrahedral or cubical elements in an attempt to capture all surfaces, but multiple passes to capture a new surface on every pass can solve the problem in a rather simple way. In addition, we do not have to explicitly provide the



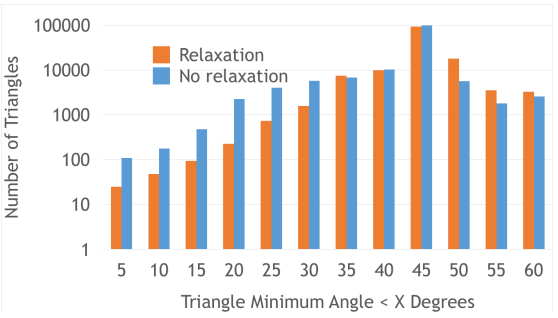
(a) Mesh Before Smoothing



(b) Mesh After Smoothing

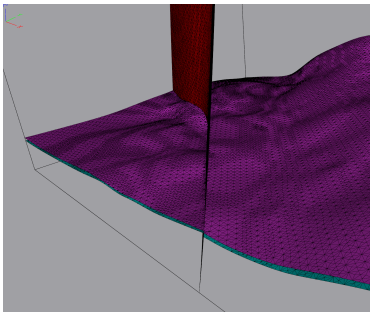


(c) Zoomed-in View

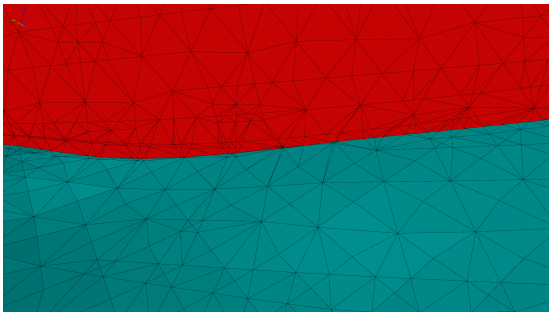


(d) Angle Histogram

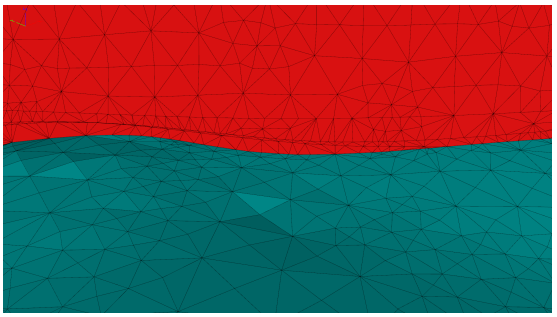
Fig. 9. Results from the Kutum domain. Two faults and horizon surfaces on either side of the faults are shown. The horizon surfaces are well separated, and hence, there are no issues meshing this domain.



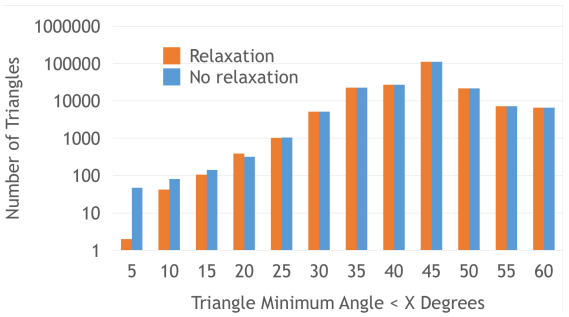
(a) Overview of the Domain



(b) Mesh at the Intersection Before Smoothing



(c) Mesh at the Intersection After Smoothing



(d) Angle Histogram

Fig. 10. Results from the Dee domain. The horizons are not planar, and hence, a refined background mesh was used to capture the geometry. A finer mesh also keeps the surfaces well separated. In our implementation, smoothing also provides a good quality surface mesh.

curve of the surface-surface intersection or the point of concurrence of several surfaces. Our algorithm is a simple, yet powerful, extension of existing techniques to capture the geometry as well to produce high-quality meshes practically.

An immediate direction of future research is to show bounds on the angles of the triangles constructed by the algorithm. It is likely that the bounds are a function of both the alpha value and the number of interacting surfaces. In order to ensure better bounds, we speculate that the alpha values vary with the number of surfaces that interact in the domain. As we have also mentioned, we need to consider the angle at which the surfaces intersect as a factor our analysis.

Our algorithm is designed for cases in which the equation of the implicit surfaces is given. In medical applications, rather than the equations of the surfaces, a scalar data field is provided for each material. The material with the largest value for its scalar field is assumed to be present at a given location. In order to generate a mesh for such an input, several modifications of the algorithm are necessary. For instance, we should first determine the surfaces that are present and where they terminate. Our technique of capturing one surface at a time is the best bet to reproduce the curves of an intersection accurately. Our future work may be focused in this direction.

Also, as has been done in [13], it should be possible to construct superior-quality meshes by constructing mesh entities in increasing order of dimensions in a bottom-up fashion. Our technique in this paper is a top-down algorithm, where we mesh the surfaces first, find the curves of the intersection next, and finally find the points of concurrence as our mesh evolves. Instead, we may first compute the points at which three or more surfaces meet, and move vertices on the background mesh at those locations and improve the quality of the mesh around them. This approach ensures that those immovable vertices do not interfere with subsequent stages. Next, we compute the edges of the surface-surface intersection, move the vertices accordingly, and improve the mesh quality. Now we have 0D and 1D elements of the mesh. Finally, we compute the triangular surface mesh in the usual fashion. This bottom-up approach is more computationally expensive than the top-down approach because additional passes on the tetrahedron are necessary to compute the 0D and 1D entities. Since we are accounting for constraints first, we may construct a better-quality mesh. This may be another direction for future work.

Implementing a GPU version of the algorithm would further improve this work by approaching a real-time meshing solution for geometry-preserving multimaterial problems. The bulk of the algorithm is root finding on large numbers of edges, which may be considered embarrassingly parallel.

Acknowledgments

The work was supported by the NIH/NIGMS Center for Integrative Biomedical Computing grant 2P41 RR0112553-12 and the DOE NET DE-EE0004449 grant. The authors would also like to thanks Christine Pickett, an editor at the University of Utah, for finding numourous typos in one of the drafts of the paper.

References

[1] S. Akkouche and E. Galin, "Adaptive implicit surface polygonization using marching triangles," *Computer Graphics Forum*, vol. 20, no. 2, pp. 67–80, 2001.

[2] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun, "Variational tetrahedral meshing," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 617–625, Jul. 2005.

[3] P. Alliez, E. C. d. Verdière, O. Devillers, and M. Isenburg, "Isotropic surface remeshing," in *Proceedings of the Shape Modeling International 2003*, ser. SMI '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 49–. [Online]. Available: http://dl.acm.org/citation.cfm?id=829510.830318

[4] J. Bloomenthal, "An implicit surface polygonizer," in *Graphics Gems IV*, P. S. Heckbert, Ed. San Diego, CA, USA: Academic Press Professional, Inc., 1994, pp. 324–349. [Online]. Available: http://dl.acm.org/citation.cfm?id=180895.180923

[5] E. Boender, W. F. Bronsvoort, and F. H. Post, "Finite-element mesh generation from constructive-solid-geometry models," *Computer-Aided Design*, vol. 26, no. 5, pp. 379 – 392, 1994.

[6] J.-D. Boissonnat, D. Cohen-Steiner, and G. Vegter, "Isotopic implicit surface meshing," in *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '04. New York, NY, USA: ACM, 2004, pp. 301–309.

[7] J.-D. Boissonnat and S. Oudot, "Provably good sampling and meshing of surfaces," *Graph. Models*, vol. 67, no. 5, pp. 405–451, Sep. 2005.

[8] J. R. Bronson, J. A. Levine, and R. T. Whitaker, *Proceedings of the 21st International Meshing Roundtable*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ch. Lattice Cleaving: Conforming Tetrahedral Meshes of Multimaterial Domains with Bounded Quality, pp. 191–209.

[9] J. R. Bronson, S. P. Sastry, J. A. Levine, and R. T. Whitaker, "Adaptive and unstructured mesh cleaving," *Procedia Engineering*, vol. 82, pp. 266 – 278, 2014, 23rd International Meshing Roundtable (IMR23).

[10] H.-L. Cheng, K. T. Dey, H. Edelsbrunner, and J. Sullivan, "Dynamic skin triangulation," *Discrete & Computational Geometry*, vol. 25, no. 4, pp. 525–568, 2001.

[11] L. P. Chew, "Guaranteed-quality mesh generation for curved surfaces," in *Proceedings of the Ninth Annual Symposium on Computational Geometry*, ser. SCG '93. New York, NY, USA: ACM, 1993, pp. 274–280.

[12] K. T. Dey and A. J. Levine, "Delaunay meshing of isosurfaces," *The Visual Computer*, vol. 24, no. 6, pp. 411–422, 2008.

[13] T. K. Dey and J. A. Levine, "Delaunay meshing of piecewise smooth complexes without expensive predicates," *Algorithms*, vol. 2, no. 4, p. 1327, 2009.

[14] Q. Du, M. D. Gunzburger, and L. Ju, "Constrained centroidal voronoi tessellations for surfaces," *SIAM Journal on Scientific Computing*, vol. 24, no. 5, pp. 1488–1506, 2003.

[15] H. Edelsbrunner and N. R. Shah, "Triangulating topological spaces," in *Proceedings of the Tenth Annual Symposium on Computational Geometry*, ser. SCG '94. New York, NY, USA: ACM, 1994, pp. 285–292.

[16] A. Gelas, S. Valette, R. Prost, and W. L. Nowinski, "Variational implicit surface meshing," *Computers & Graphics*, vol. 33, no. 3, pp. 312 – 320, 2009, IEEE International Conference on Shape Modelling and Applications 2009 .

[17] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 339–346, Jul. 2002.

[18] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature sensitive surface extraction from volume data," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 57–66.

[19] F. Labelle and J. R. Shewchuk, "Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.

[20] X. Liang and Y. Zhang, "An octree-based dual contouring method for triangular and tetrahedral mesh generation with guaranteed angle range," *Engineering with Computers*, vol. 30, no. 2, pp. 211–222, 2014.

[21] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Aug. 1987.

[22] M. Meyer, R. Whitaker, R. M. Kirby, C. Ledergerber, and H. Pfister, "Particle-based sampling and meshing of surfaces in multimaterial volumes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1539–1546, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2008.154

[23] S. Oudot, L. Rineau, and M. Yvinec, *Proceedings of the 14th International Meshing Roundtable*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ch. Meshing Volumes Bounded by Smooth Surfaces, pp. 203–219.

[24] J. Schreiner, C. Scheidegger, and C. Silva, "High-quality extraction of isosurfaces from regular and irregular grids," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1205–1212, Sept 2006.

[25] R. Shu, C. Zhou, and M. S. Kankanhalli, "Adaptive marching cubes," *The Visual Computer*, vol. 11, no. 4, pp. 202–217, 1995.

[26] B. T. Stander and J. C. Hart, "Guaranteeing the topology of an implicit surface polygonization for interactive modeling," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 279–286.

[27] Y. Zhang and J. Qian, "Dual contouring for domains with topology ambiguity," *Computer Methods in Applied Mechanics and Engineering*, vol. 217220, pp. 34 – 45, 2012.

[28] Z. Zhong, X. Guo, W. Wang, B. Lvy, F. Sun, Y. Liu, and W. Mao, "Particle-based anisotropic surface meshing," *ACM Transactions on Graphics (SIGGRAPH conference proceedings)*, 2013.